DOI:
10.1007/978-3-030-39881-1_7

Document Version
Peer reviewed version

Shortest Covers of All Cyclic Shifts of a String

Maxime Crochemore[1][0000−0003−1087−1419],
Costas S. Iliopoulos[1][0000−0003−3909−0077],
Jakub Radoszewski[2⋆][0000−0002−0067−6401],
Wojciech Rytter[2][0000−0002−9162−6724], Juliusz
Straszyński[2,⋆][0000−0003−2207−0053], Tomasz Waleń[2][0000−0002−7369−3309],⋆, and
Wiktor Zuba[2][0000−0002−1988−3507],⋆

[1] Department of Informatics, King's College London, London, UK,
{maxime.crochemore,c.iliopoulos}@kcl.ac.uk
[2] Institute of Informatics, University of Warsaw, Warsaw, Poland
{jrad,rytter,jks,walen,w.zuba}@mimuw.edu.pl

Abstract. A factor W of a string X is called a cover of X, if X can be constructed by concatenations and superpositions of W. Breslauer (IPL, 1992) proposed a well-known $\mathcal{O}(n)$-time algorithm that computes the shortest cover of every prefix of a string of length n. We show an $\mathcal{O}(n \log n)$-time algorithm that computes the shortest cover of every cyclic shift of a string and an $\mathcal{O}(n)$-time algorithm that computes the shortest among these covers. A related problem is the number of different lengths of shortest covers of cyclic shifts of the same string of length n. We show that this number is $\Omega(\log n)$.

1 Introduction

We consider strings as finite sequences of letters from an integer alphabet Σ. The notion of periodicity in strings and its many variants have been well-studied in many fields like combinatorics on words, pattern matching, data compression, automata theory, formal language theory, and molecular biology. A typical regularity, the *period* U of a given string X, grasps the repetitiveness of X since X is a prefix of a string constructed by concatenations of U. If $X = AWB$, for some, possibly empty, strings A, W, B, then W is called a *factor* of X and, respectively, X is a superstring of W. A factor W of X is called a *cover* of X, if X can be constructed by concatenations and superpositions of W. A factor W of X is called a *seed* of X, if there exists a superstring of X which is constructed by concatenations and superpositions of W. For example, *abc* is a period of *abcabcabca*, *abca* is a cover of *abcabcaabca*, and *abca* is a seed of *bcabcaabc*. The notions "cover" and "seed" are generalizations of periods in the sense that superpositions as well as concatenations are considered to define them, whereas only concatenations are considered for periods.

In computation of covers, two problems have been considered in the literature. The shortest-cover problem (also known as the superprimitivity test) is that of

⋆ Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

computing the shortest cover of a given string of length n, and the all-covers problem is that of computing all the covers of a given string. Apostolico et al. [1] introduced the notion of covers and gave a linear-time algorithm for the shortest-cover problem. Breslauer [4] proposed an on-line algorithm for computing the shortest cover that works in linear time. In particular, his algorithm computes the shortest cover of every prefix of a string. The other direction was taken by Moore and Smyth [20,21] and by Li and Smyth [19] who computed all the covers of a string and a representation of all the covers of all prefixes of a string, respectively.

Covers of circular strings were also considered. It is implicit in [13] that covers of a circular string S are exactly seeds of S^2 (see also [15]).

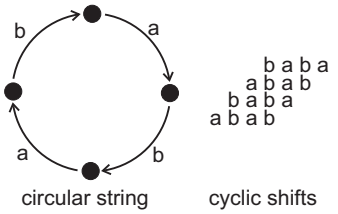

Fig. 1. The string *aba* is a cover of the string $S = abab$ treated as a single circular string, but is not a cover of any of cyclic shifts of S.

All the seeds of a string of length n can be represented in $\mathcal{O}(n)$ space as a collection of a linear number of disjoint paths in the suffix trees of the string and of its reversal. This representation can be computed in $\mathcal{O}(n \log n)$ time [13] and even in $\mathcal{O}(n)$-time [16]. Recently it was also shown in [17] that all the seeds can also be represented as a linear number of disjoint paths in just the suffix tree of the string. This implies the following fact:

Lemma 1. *The problem of computing the shortest cover of a circular string can be solved in linear time.*

We say that a string Y is a *cyclic shift* of a string X if $X = AB$ and $Y = BA$ for some strings A and B; in this case we also write $Y = rot_{|A|}(X)$. It seems that the problem of computing shortest covers of all cyclic shifts of a string is harder than that of computing the shortest cover of a circular string. A straightforward application of any of the aforementioned algorithms for computing covers of a string yields an $\mathcal{O}(n^2)$-time solution to the problem. One should note that covers of circular strings are a different notion than that of covers of cyclic shifts of a string; see Fig. 1.

The shortest covers of cyclic shifts of a string can behave rather irregularly. For example, the length of the shortest cover of $S = abaababababababaaba$ equals 3, whereas the shortest cover of $rot_1(S)$ has length 18.

We consider the following problem.

Shortest Covers of All Cyclic Shifts of a String
Input: A string S of length n.
Output: The lengths of the shortest covers of all cyclic shifts of S.

Let S be a string of length n and $ShCov(S)$ denote the shortest cover of S. We introduce an array $CyCo_S$ of length n such that $CyCo_S[i] = |ShCov(rot_i(S))|$. Our main result is computing this array. We also denote

$$CyCoSet(S) = \{CyCo_S[i] \,:\, i = 0, \ldots, n-1\}.$$

Example 2. For the Fibonacci strings $S_1 = abaab$, $S_2 = abaababaabaab$ we have:

$$CyCo_{S_1} = [5,5,5,3,5],\ CyCo_{S_2} = [5,5,13,3,...]$$

$$CyCoSet(S_1) = \{3,5\},\ CyCoSet(S_2) = \{3,5,8,13\}.$$

Our results. We show that the whole array $CyCo_S$ and $\min_i CyCo_S[i]$ for a string S of length n can be computed in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ time, respectively. For this we use a characterization of covers of cyclic shifts of a string by seeds and squares, i.e., strings of the form W^2, and the suffix tree data structure. We also show that there exists a (known) infinite family of strings for which $|CyCoSet(S)| = \Theta(\log |S|)$.

Structure of the paper. In Section 2 we recall the definition and basic properties of a suffix tree of a string. Then in Section 3 we present characterizations of shortest covers of cyclic shifts of a string, which lead us to the main algorithmic results in Section 4. The lower bound on the size of the *CyCoSet* set is shown using Fibonacci strings in Section 5. We conclude and mention some open problems in Section 6.

2 Applications of the Suffix Tree

Recall that a *suffix tree* of a string S is a compact trie of all the suffixes of $S\#$, where $\#$ is a special end marker. The root, branching nodes, and leaves of the tree are *explicit*. All the remaining nodes are *implicit* in the tree. Each leaf is labeled with the starting position of the corresponding suffix. Every factor of S is represented as an explicit or implicit node of the tree. A suffix tree of a string of length n over an integer alphabet can be constructed in $\mathcal{O}(n)$ time [10].

Observation 1 *Let S be a string of length n. After $\mathcal{O}(n)$-time preprocessing, all the occurrences of a factor of S, represented as a node in the suffix tree of S, can be reported in linear time w.r.t. the number of these occurrences.*

Proof. It suffices to store a list L of leaves of the suffix tree in a left-to-right order. Then for every explicit node v of the tree, we precompute the endpoints of the sublist of L that corresponds to the occurrences of the string v. This precomputation is done bottom-up in $\mathcal{O}(n)$ time. □

We also use the following lemma.

Lemma 3 ([17]).
Given a collection of factors $U_1, \ldots, U_k$ of a string S of length n, each represented by an occurrence in S, in $\mathcal{O}(n+k)$ time we can compute the implicit or explicit node in the suffix tree of S that corresponds to each factor U_i. Moreover, all these nodes can be made explicit in $\mathcal{O}(n+k)$ time.

The set (possibly of a quadratic size) of all seeds of a string can be represented as a collection of linearly many disjoint paths in the suffix tree [17]. It can be assumed that each path belongs to a single edge of the suffix tree. The endpoints of the paths can be implicit nodes. For an example, see Fig. 2.

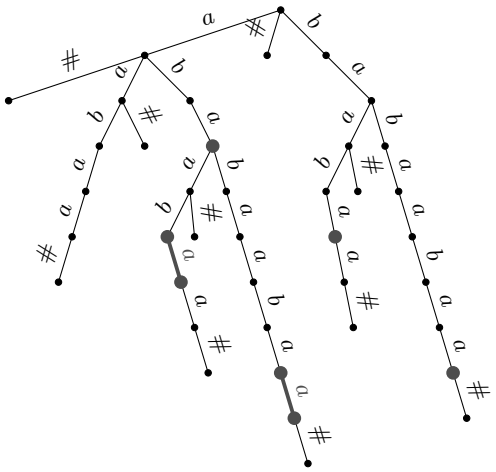

Fig. 2. The string $S = ababaabaa$ has the following seeds: *aba*, *abaab*, *baaba*, *abaaba*, *ababaaba*, *babaabaa*, *ababaabaa*. They can be represented on the (uncompressed) suffix tree of S as shown in the figure. Each seed is a path from root to marked node. In some cases, e.g. *abaab*, *abaaba*, multiple seeds are represented on a single path.

3 Covers of Cyclic Shifts

A string X is called *primitive* if $X = Y^k$ for positive integer k implies that $k = 1$. A string of the form Z^2 is called a *square*; it is called *primitively rooted* if Z

is primitive. We denote by $Squares(S)$ the set of factors Z of S such that the square Z^2 is also a factor of S and by $PSquares(S)$ the subset of $Squares(S)$ that consists only of primitive strings. We further denote by $Seeds(S)$ the set of factors which are seeds of S. We use these sets of S^3 in order to characterize covers of all cyclic shifts of S.

Lemma 4. *Let S be a string of length n and C be a string of length up to n. Then C is a cover of $rot_i(S)$ if and only if $C \in Seeds(S^3) \cap Squares(S^3)$ and C^2 occurs with its center at position $j \equiv i \pmod n$ in S^3.*

Moreover, if C is the shortest cover of $rot_i(S)$, then $C \in Seeds(S^3) \cap PSquares(S^3)$.

Proof.

($\Rightarrow$) String C is a cover of $(rot_i(S))^4$, and thus a seed of its factor S^3. Moreover, $S^3[j - |C|, j + |C| - 1]$, that is, the factor of S^3 of length $2|C|$ with center at position j, is equal to C^2 for $j = i + n$.

($\Leftarrow$) The square C^2 occurs in S^3 with its center at position $j \equiv i \pmod n$. Thus C is a prefix and a suffix of $rot_j(S) = rot_i(S)$ as $|C| < n$. C is also a seed of $rot_i(S)$ which is a factor of S^3, hence it is a cover of $rot_i(S)$.

As for the "moreover" part, it suffices to note that the shortest cover of a string is obviously primitive. □

Example 5. In the above lemma, one could not take S^2 instead of S^3. Indeed, for $S = abaaaaba$ we have that $rot_4(S) = aabaabaa$ has the shortest cover $aabaa$, but $S^2 = abaaaabaabaaaaba$ does not contain the square $(aabaa)^2$.

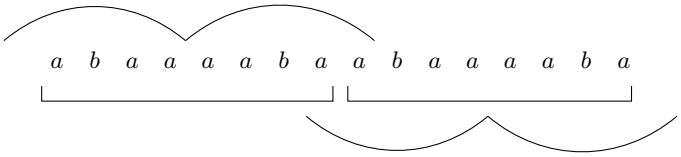

Fig. 3. Illustration of Example 5.

Let $T(S^3)$ be the suffix tree of S^3 in which we distinguish the nodes v corresponding to strings Z^2 for $Z \in Seeds(S^3) \cap PSquares(S^3)$. These nodes are called *candidate nodes*. Some of these nodes could be implicit nodes in the suffix tree. Then they are made explicit. Denote by $CandAnc(v)$ the set of ancestor nodes of v in $T(S^3)$ which are candidate nodes. Let $|v|$ be the length of the string corresponding to the node v.

We can reformulate Lemma 4 as follows:

Lemma 6. *$CyCo_S[i]$, i.e., the length of the shortest cover of $rot_i(S)$, equals*

$$\min_{j,v} \{\, k \,:\, k = |v|/2,\ i = (j + k) \bmod n,\ j \in Leaves(T(S^3)),\ v \in CandAnc(j)\}.$$

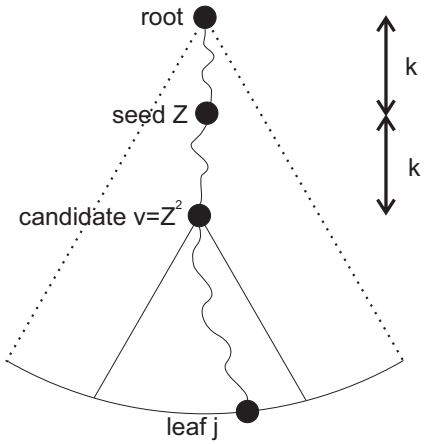

Fig. 4. Illustration of Lemma 6. The situation when $CyCo_S[i] = k$. We have that $i = (j + k) \bmod n$ and Z^2 is a primitively rooted square of length $2k$; it corresponds to the node v which is possibly inside an edge of the suffix tree.

Clearly if C is a cover of $rot_i(S)$, then C is a cover of S treated as a circular string. As we have already noted in Fig. 1, the converse is not necessarily true. However, we show that every shortest cover of the circular string S is a cover of the corresponding cyclic shift of S.

Lemma 7. *A shortest cover of a circular string is always a (shortest) cover of some cyclic shift.*

Proof. We need the following claim.

Claim (See [13]). String C is a cover of S considered as a circular string iff it is a seed of S^2, hence also iff it is a seed of S^3.

Consider a cover C of a circular string S, such that C^2 does not occur in it. Consider the last position covered by any occurrence of C in the string.

The position must be also covered by another occurrence of C (the next position must be covered and cannot be the first position of some C). Thus by erasing the last position of C we obtain a shorter cover. Hence if C is a shortest cover then C^2 must appear in the circular string S.

By Lemma 4 it is a cover of some cyclic shift of the string. □

By computing the shortest cover of the circular string S using Lemma 1 we obtain the following preliminary result.

Corollary 8. *For a string S of length n,* $\min CyCo_S$ *can be computed in $\mathcal{O}(n)$ time.*

4 Main Algorithm

First we have to show how to compute efficiently the tree $T(S^3)$. We denote by $OccPSquares(S)$ the set of all occurrences of primitively rooted squares in S. Each occurrence is represented in $\mathcal{O}(1)$ space as a factor of S. A direct consequence of the Three-square-prefix Lemma, see [8], is that a string of length n has no more than $\log n$ prefixes that are primitively rooted squares.

Lemma 9 ([8]). *For a string S of length n, $|OccPSquares(S)| = \mathcal{O}(n \log n)$.*

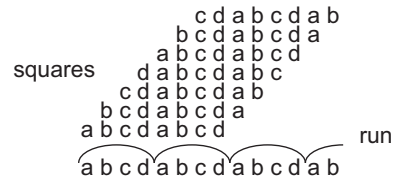

Fig. 5. Primitive squares can be derived from runs (maximal repetitions), knowing the shortest periods of runs.

Lemma 10. *For a string S of length n, $|PSquares(S)| = \mathcal{O}(n)$ and this set can be computed in $\mathcal{O}(n)$ time.*

Proof. Let us start with efficient computation of squares.

Claim ([7,9,11,12]).
For a string S of length n, $|Squares(S)| = \mathcal{O}(n)$ and this set can be computed in $\mathcal{O}(n)$ time.

By the claim, $|PSquares(S)| = \mathcal{O}(n)$ since $PSquares(S) \subseteq Squares(S)$.

The set $PSquares(S)$ can be computed by filtering out the factors from $Squares(S)$ that are not primitive. This can be done in $\mathcal{O}(1)$ time per factor after $\mathcal{O}(n)$-space and time preprocessing using so-called Two-Period queries [3,18]. A more direct approach would be to (effortlessly) adapt the algorithm for computing different square factors from [7] using relations between primitive squares and *runs* (maximal repetitions); see Figure 5. □

Lemma 11. *The tree $T(S^3)$ can be computed in $\mathcal{O}(n)$ time.*

Proof. We use a version of a minimal augmented suffix tree (MAST, in short), a data structure that was initially introduced in [2].

Let us recall that Lemma 3 can be used to augment the suffix tree with nodes that correspond to a set of factors of S^3. We first apply the lemma to the collection of factors $PSquares(S^3)$, which can be efficiently computed due to the previous lemma.

Then we compute a representation of all the seeds of S^3 in the suffix tree using the algorithm from [17]. The representation consists of a collection of disjoint paths, each located on a single edge in the suffix tree; see Figure 2. The endpoints of the paths that are implicit nodes can also be made explicit using the lemma.

For every node v corresponding to an element in $\mathit{PSquares}(S^3)$, we check if it is located on some path that belongs to the representation of $\mathit{Seeds}(S^3)$. Finally, we again use Lemma 3 for the original suffix tree of S^3 and set of factors Z^2 that correspond to all such elements $Z \in \mathit{PSquares}(S^3) \cap \mathit{Seeds}(S^3)$ to obtain the set of candidate nodes. This completes the proof. □

The number of integers $(j + k)$ equal *modulo* n is constant, hence we can forget about computing modulo n and for each $0 \leq i < 3n$ we are to compute:

$$\min_{j,v} \{\, k \;:\; k = |v|/2,\ i = j + k,\ j \in \mathit{Leaves}(T(S^3)),\ v \in \mathit{CandAnc}(j)\}. \qquad (1)$$

Algorithm ComputeCyCo

1. Initialize each entry of CyCo to $+\infty$
2. Compute $T(S^3)$
3. **for each** candidate node v in $T(S^3)$ **do**
   for each occurrence $S^3[j, j + |v| - 1]$ of v in S^3 **do**
   $i := (j + |v|) \bmod n$
   $\mathit{CyCo}[i] := \min(\mathit{CyCo}[i], |v|)$
4. **return** CyCo

Theorem 12. *The algorithm* ComputeCyCo *computes the lengths of shortest covers for all cyclic shifts of a string in* $\mathcal{O}(n \log n)$ *time.*

Proof. In the third paragraph we simply implement (1). This proves correctness. The occurrences of a node are computed using Observation 1. The required complexity follows from Lemma 9 and Lemma 11. □

5 Strings with Arbitrarily Large Size of $\mathit{CyCoSet}(S)$

We show that the size of $\mathit{CyCoSet}(S)$, for a binary alphabet, is not bounded by a constant. It grows at least logarithmically.

Recall that the Fibonacci strings are defined as $\mathit{Fib}_0 = b$, $\mathit{Fib}_1 = a$, $\mathit{Fib}_k = \mathit{Fib}_{k-1}\mathit{Fib}_{k-2}$ for $k \geq 2$. In other words, $\mathit{Fib}_k = \phi^k(\mathit{Fib}_0)$, where ϕ is a morphism

$$\phi(a) = ab, \quad \phi(b) = a.$$

Hence

$$\mathit{Fib}_2 = ab,\ \mathit{Fib}_3 = aba,\ \mathit{Fib}_4 = abaab,\ \mathit{Fib}_5 = abaababa, \ldots.$$

We denote $F_k = |Fib_k|$, the k-th Fibonacci number.

We use the following well known properties of Fibonacci strings.

Observation 2 *Fib_k does not contain the factors aaa and bb.*

Fact 1 (see [22,14]) *For every non-empty factor U^2 of Fib_k, U is a cyclic shift of Fib_m for some m.*

Fib_2

cyclic shift	shortest cover	length
ab	ab	2
ba	ba	2

Fib_3

aba	aba	3
baa	baa	3
aab	aab	3

Fib_4

abaab	abaab	5
baaba	baaba	5
aabab	aabab	5
ababa	aba	3
babaa	babaa	5

Fib_5

abaababa	aba	3
baababaa	baababaa	8
aababaab	aababaab	8
ababaaba	aba	3
babaabaa	babaabaa	8
abaabaab	abaab	5
baabaaba	baaba	5
aabaabab	aabaabab	8

Fib_6

cyclic shift	shortest cover	length
abaababaabaab	abaab	5
baababaabaaba	baaba	5
aababaabaabab	aababaabaabab	13
ababaabaababa	aba	3
babaabaababaa	babaabaababaa	13
abaabaababaab	abaab	5
baabaababaaba	baaba	5
aabaababaabab	aabaababaabab	13
abaababaababa	aba	3
baababaababaa	baababaa	8
aababaababaab	aababaab	8
ababaababaaba	aba	3
babaababaabaa	babaababaabaa	13

Fig. 6. Shortest covers of cyclic shifts of Fibonacci strings.

An example for the theorem below can be found in Fig. 6.

Theorem 13. *For $k \geq 3$, $CyCoSet(Fib_k) = \{F_3, \ldots, F_k\}$.*

Proof. We show two inclusions.

Proof of the inclusion $\{F_3, \ldots, F_k\} \supseteq CyCoSet(Fib_k)$.

By Lemma 4, every element of the set $CyCoSet(Fib_k)$ is a square half of length at most F_k in Fib_k^3. By Fact 1, the lengths of square halves in a Fibonacci string are Fibonacci numbers. It suffices to note that, for $k \geq 3$, Fib_k^3 is a factor of Fib_{k+5} since

$$Fib_8 = abaababaabaababaab\underline{abaabaaba}baabaab$$

contains a cube Fib_3^3 (underlined). It can be readily verified that no string of length $F_1 = 1$ and $F_2 = 2$ covers Fib_k for $k \geq 3$.

Proof of the inclusion $\{F_3, \ldots, F_k\} \subseteq CyCoSet(Fib_k)$.

We will prove that for every $i = 3, \ldots, k$, there exists a cyclic shift S of Fib_k such that $|ShCov(S)| = F_i$ and $S \neq aaXbab$ for a binary string X. The proof goes by induction over k. For $k = 3$ the conclusion is straightforward. Assume now that the conclusion holds for $k - 1$.

Let us consider $i \in \{3, \ldots, k-1\}$ and let S be a cyclic shift of Fib_{k-1} such that $|ShCov(S)| = F_i$ and S is not of the form $aaXbab$. We use the following observation.

Observation 3 *Let $S_{1,1} = abXb$, $S_{1,2} = bXba$, $S_{2,1} = baaXaa$, and $S_{2,2} = aaXaab$ be cyclic shifts of a Fibonacci string, where X is a binary string. For every $i = 1, 2$ and string C, C is a cover of $S_{i,1}$ if and only if $rot_1(C)$ is a cover of $S_{i,2}$.*

Proof. Any cover C of $S_{1,1}$ starts with the letter a and ends with the letter b. By Observation 2, each of its occurrences except for the occurrence as a suffix is followed by the letter a. Hence, $rot_1(C)$ is a cover of $S_{1,2}$. Similarly, a cover C' of $S_{1,2}$ starts with the letter b and ends with the letter a, so each of its occurrences except for the occurrence as a prefix is preceded by the letter a. Hence, $rot_{|C'|-1}(C')$ is a cover of $S_{1,1}$.

The proof for $S_{2,1}$ and $S_{2,2}$ is analogous. □

If S ends with the letter b, we either move its first letter to its end to obtain a string that matches $S_{1,2}$ or the last letter to the start to obtain a string that matches $S_{2,1}$. We use the additional property that S is not of the form $aaXbab$, in which case none of the shifts would be possible. After the potential transformation, $|ShCov(S)|$ did not change and S starts with the letter b.

Now $S' = \phi(S)$ is a cyclic shift of Fib_k that ends with $\phi(a) = ab$.

Observation 4 *Assume that $S' = \phi(S)$ and S' ends with the letter b. Let C' be a cover of S'. Then there exists a unique cover C of S such that $\phi(C) = C'$.*

By the observation, if $C = ShCov(S)$, then $C' = \phi(C)$ is the shortest cover of S'. By Lemma 4, C^2 occurs in Fib_{k-1}^3. Hence, Fact 1 implies that C is a cyclic shift of Fib_i, so C' is a cyclic shift of Fib_{i+1} and $|C'| = F_{i+1}$.

Thus we have obtained that $\{F_4, \ldots, F_k\} \subseteq CyCoSet(Fib_k)$. To conclude, we notice that $rot_3(Fib_k)$ starts and ends with $Fib_2 = aba$, so aba is its cover by Observation 2 (and the shortest cover by the first inclusion). Thus $F_3 \in CyCoSet(Fib_k)$.

Consequently, $\{F_3, F_4, \ldots, F_k\} \subseteq CyCoSet(Fib_k)$. □

6 Conclusions and Open Problems

Breslauer [4] proposed a linear-time algorithm for computing the shortest cover of every prefix of a string. We have proposed an $\mathcal{O}(n \log n)$-time algorithm for computing the shortest cover of every cyclic shift of a string. It remains an open problem if these values can be computed in $\mathcal{O}(n)$ time.

$\mathcal{O}(n)$, $\mathcal{O}(n \log n)$ and $\mathcal{O}(n^2)$-time algorithms for computing the shortest left seed, right seed, and seed, respectively, of all the prefixes of a string are known; see [5,6]. Here left and right seed are notions that are intermediate between cover and seed. It remains an open problem if the shortest left seed, right seed, and seed can be computed efficiently for all the cyclic shifts of a string.

Based on computer experiments we make the following conjecture.

Conjecture 14. For a string S of length n, $|CyCoSet(S)| = \mathcal{O}(\log n)$.

References

1. Apostolico, A., Farach, M., Iliopoulos, C.S.: Optimal superprimitivity testing for strings. Inf. Process. Lett. **39**(1), 17–20 (1991). https://doi.org/10.1016/0020-0190(91)90056-N
2. Apostolico, A., Preparata, F.P.: Data structures and algorithms for the string statistics problem. Algorithmica **15**(5), 481–494 (1996). https://doi.org/10.1007/BF01955046
3. Bannai, H., I, T., Inenaga, S., Nakashima, Y., Takeda, M., Tsuruta, K.: The "runs" theorem. SIAM J. Comput. **46**(5), 1501–1514 (2017). https://doi.org/10.1137/15M1011032
4. Breslauer, D.: An on-line string superprimitivity test. Inf. Process. Lett. **44**(6), 345–347 (1992). https://doi.org/10.1016/0020-0190(92)90111-8
5. Christou, M., Crochemore, M., Guth, O., Iliopoulos, C.S., Pissis, S.P.: On left and right seeds of a string. J. Discrete Algorithms **17**, 31–44 (2012). https://doi.org/10.1016/j.jda.2012.10.004
6. Christou, M., Crochemore, M., Iliopoulos, C.S., Kubica, M., Pissis, S.P., Radoszewski, J., Rytter, W., Szreder, B., Waleń, T.: Efficient seed computation revisited. Theor. Comput. Sci. **483**, 171–181 (2013). https://doi.org/10.1016/j.tcs.2011.12.078
7. Crochemore, M., Iliopoulos, C.S., Kubica, M., Radoszewski, J., Rytter, W., Waleń, T.: Extracting powers and periods in a word from its runs structure. Theor. Comput. Sci. **521**, 29–41 (2014). https://doi.org/10.1016/j.tcs.2013.11.018
8. Crochemore, M., Rytter, W.: Squares, cubes, and time-space efficient string searching. Algorithmica **13**(5), 405–425 (1995). https://doi.org/10.1007/BF01190846
9. Deza, A., Franek, F., Thierry, A.: How many double squares can a string contain? Discrete Applied Mathematics **180**, 52–69 (2015). https://doi.org/10.1016/j.dam.2014.08.016
10. Farach, M.: Optimal suffix tree construction with large alphabets. In: 38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997. pp. 137–143. IEEE Computer Society (1997). https://doi.org/10.1109/SFCS.1997.646102
11. Fraenkel, A.S., Simpson, J.: How many squares can a string contain? J. Comb. Theory, Ser. A **82**(1), 112–120 (1998). https://doi.org/10.1006/jcta.1997.2843

12. Gusfield, D., Stoye, J.: Linear time algorithms for finding and representing all the tandem repeats in a string. J. Comput. Syst. Sci. **69**(4), 525–546 (2004). https://doi.org/10.1016/j.jcss.2004.03.004
13. Iliopoulos, C.S., Moore, D.W.G., Park, K.: Covering a string. Algorithmica **16**(3), 288–297 (1996). https://doi.org/10.1007/BF01955677
14. Iliopoulos, C.S., Moore, D.W.G., Smyth, W.F.: A characterization of the squares in a Fibonacci string. Theor. Comput. Sci. **172**(1-2), 281–291 (1997). https://doi.org/10.1016/S0304-3975(96)00141-7
15. Iliopoulos, C.S., Moore, D.W.G., Smyth, W.F.: The covers of a circular Fibonacci string. Journal of Combinatorial Mathematics and Combinatorial Computing **26**, 227–236 (1998)
16. Kociumaka, T., Kubica, M., Radoszewski, J., Rytter, W., Waleń, T.: A linear time algorithm for seeds computation. In: Rabani, Y. (ed.) Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012. pp. 1095–1112. SIAM (2012). https://doi.org/10.1137/1.9781611973099.86
17. Kociumaka, T., Kubica, M., Radoszewski, J., Rytter, W., Waleń, T.: A linear time algorithm for seeds computation. CoRR **abs/1107.2422v2** (2019), `http://arxiv.org/abs/1107.2422v2`
18. Kociumaka, T., Radoszewski, J., Rytter, W., Waleń, T.: Internal pattern matching queries in a text and applications. In: Indyk, P. (ed.) Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015. pp. 532–551. SIAM (2015). https://doi.org/10.1137/1.9781611973730.36
19. Li, Y., Smyth, W.F.: Computing the cover array in linear time. Algorithmica **32**(1), 95–106 (2002). https://doi.org/10.1007/s00453-001-0062-2
20. Moore, D.W.G., Smyth, W.F.: An optimal algorithm to compute all the covers of a string. Inf. Process. Lett. **50**(5), 239–246 (1994). https://doi.org/10.1016/0020-0190(94)00045-X
21. Moore, D.W.G., Smyth, W.F.: A correction to "An optimal algorithm to compute all the covers of a string". Inf. Process. Lett. **54**(2), 101–103 (1995). https://doi.org/10.1016/0020-0190(94)00235-Q
22. Séébold, P.: Propriétés combinatoires des mots infinis engendrés par certains morphismes. Report no. 85-16, LITP, Paris (1985)